Solving large systems on HECToR using the 2-lagrange multiplier methods

Solving large systems on HECToR using the 2-Lagrange multiplier methods

Anastasios Karangelis[1], Sébastien Loisel[1], and Chris Maynard[2]

1 Introduction

We consider the model problem,

$$-\Delta\tilde{u} = \tilde{f} \text{ in } \Omega \text{ and } \tilde{u} = 0 \text{ on } \partial\Omega. \tag{1}$$

In order to solve the problem numerically we discretize it by some suitable method[1] and as a result we get the system,

$$Au = f, \tag{2}$$

where A is a large symmetric and positive definite sparse matrix, f is the load vector and u is the desired discrete solution of our problem. Note that we use the notation $\tilde{u} = \tilde{u}(x)$ for the solution $\tilde{u} \in H_0^1(\Omega)$ and u for corresponding finite element coefficient vector.

We decompose our square model domain Ω into nonoverlapping rectangular subdomains $\Omega_1, \ldots, \Omega_p$ and we define the artificial interface $\Gamma = \Omega \cap (\bigcup_{i=1}^p \partial\Omega_i)$, such that $\Omega = \Gamma \cup \left(\bigcup_{k=1}^p \Omega_k\right)$ with disjoint unions. Although our numerical experiments are on a square, the analysis in [3, 5] applies to more general "shape-regular" domain decompositions and grids such as described in [8].

The local Robin subproblems are,

$$\begin{cases} -\Delta\tilde{u}_k = \tilde{f} & \text{in } \Omega_k, \\ \tilde{u}_k = 0 & \text{on } \partial\Omega_k \cap \partial\Omega, \\ (a + D_\nu)\tilde{u}_k = \tilde{\lambda}_k & \text{on } \partial\Omega_k \cap \Gamma; \end{cases} \tag{3}$$

where $a > 0$ is the Robin parameter, $k = 1, \ldots, p$ and D_ν denotes the directional derivative in the direction of the unit outwards normal vector ν of $\partial\Omega$, and $\tilde{\lambda}_k$ is the Robin data imposed on the "artificial interface" $\partial\Omega_k \cap \Gamma$.

We now discretize system (3) using the finite element method. This leads to linear systems of the form,

$$\begin{bmatrix} A_{Ik} & A_{I\Gamma k} \\ A_{\Gamma Ik} & A_{\Gamma\Gamma k} + aI \end{bmatrix} \begin{bmatrix} u_{Ik} \\ u_{\Gamma k} \end{bmatrix} = \begin{bmatrix} f_{Ik} \\ f_{\Gamma k} \end{bmatrix} + \begin{bmatrix} 0 \\ \lambda_k \end{bmatrix}. \tag{4}$$

[1] Dept. of Mathematics, Heriot-Watt University, Edinburgh EH14 4AS, United Kingdom, e-mail: {ak411}{S.Loisel}@hw.ac.uk .[2] EPCC, University of Edinburgh, Edinburgh EH9 3JZ, United Kingdom, e-mail: c.maynard@ed.ac.uk and Met Office, FitzRoy Road, Exeter, EX1 3PB, e-mail: christopher.maynard@metoffice.gov.uk

[1] In general this could be by finite elements or finite differences.

Here, the subscript I denotes nodes that are Interior to Ω_k, while the subscript Γ denotes nodes on $\Gamma \cap \partial\Omega_k$; this notation is consistent with existing literature, see [5], [8]. Using a Schur complement, we eliminate the interior nodes of equation (4) to get the equivalent system,

$$(S+aI)u_G = g+\lambda, \tag{5}$$

where $S = \mathrm{diag}\{S_1,\ldots,S_p\}$ with symmetric and semidefinite Schur complements $S_k = A_{\Gamma\Gamma k} - A_{\Gamma Ik}A_{Ik}^{-1}A_{I\Gamma k}$; the column vector $u_G = [u_{\Gamma 1}^T,\ldots,u_{\Gamma p}^T]^T$ is the multi-valued trace (with one value per interface vertex per adjacent subdomain), the Robin data are $\lambda = [\lambda_1^T,\ldots,\lambda_p^T]^T$ and the "accumulated fluxes" are $g_k = f_{\Gamma k} - A_{\Gamma Ik}A_{Ik}^{-1}f_{Ik}$.

We define the scaled "Robin-to-Dirichlet" map $Q = \mathrm{diag}\{Q_1,\ldots,Q_p\}$, where $Q_k = a(S_k+aI_k)^{-1}$ and (5) can be rewritten as,

$$au_G = Q(g+\lambda). \tag{6}$$

The multi-valued trace u_G can be interpreted as the multi-valued trace of a finite element function $\tilde{u}(x)$ which has jump discontinuities along Γ. For each vertex $x_j \in \Gamma$ on the interface, we define m_j to be the number of subdomains adjacent to x_j. A vertex with $m_j = 2$ is called a regular interface point while a vertex with $m_j > 2$ is called a cross point. The solution of (1) is continuous and so we must impose continuity on $\tilde{u}(x)$ (or equivalently, on its finite element trace vector u_G). To that end, we define K to be the orthogonal projection matrix which averages the function values for each interface vertex x_j; note that the range of K is precisely the space of continuous many-sided traces. Hence, u_G is continuous if and only if,

$$Ku_G = u_G. \tag{7}$$

Additionally we require the "fluxes" to match which is equivalent to

$$K(Su_G) = Kg. \tag{8}$$

1.1 Obtaining the S2LM and 2LM systems

From (5) and (7) we get that,

$$KQ(\lambda+g) = Q(\lambda+g), \tag{9}$$

and from (8) we get,

$$K(g+\lambda-Q(g+\lambda)) = Kg. \tag{10}$$

We add (9) and (10) to get the **symmetric 2-Lagrange multiplier system** (S2LM) ,

$$(Q-K)\lambda = -Qg. \tag{11}$$

Multiplying both sides of (11) on the left by $(I-2K)$, we get the corresponding **nonsymmetric 2-Lagrange system** (2LM),

$$(I-2K)(Q-K)\lambda = -(I-2K)Qg. \tag{12}$$

We now briefly summarize some known results about the 2-Lagrange methods and refer to [4], [5], [3] for details.

Theorem 1. *We define E to be the orthogonal projection onto the kernel of S. Assume that $\|EK\| < 1$. Then (11) is equivalent to (2).*

Theorem 2. *[4] The nonsymmetric 2-Lagrange system, $(I-2K)(Q-K)$ is an Optimised Schwarz Method (at least for two subdomains)*

The 2-Lagrange multiplier methods also have a coarse grid preconditioner,

$$P = I - EKE, \tag{13}$$

leading to the 2-level methods,

$$P^{-1}(Q-K)\lambda = -P^{-1}Qg, \tag{14}$$
$$P^{-1}(I-2K)(Q-K)\lambda = -P^{-1}(I-2K)Qg. \tag{15}$$

Theorem 3. *The optimized Robin parameter $a = \sqrt{s_{min}s_{max}}$, where s_{min} and s_{max} are the extremal eigenvalues of S. Moreover,*
The condition number for the 1-level methods is $O(h^{-1/2}H^{-3/2})$
The condition number for the 2-level methods is $O(H/h)^{1/2}$

2 Implementation of symmetric and nonsymmetric 2-Lagrange multiplier and large scale experiments on HECToR

The numerical experiments were run on HECToR, a Cray XE6 with 2816 compute nodes each comprising of two 16-core AMD Opeteron Interlagos processors. Each of the 16-core socket is coupled with a Cray Gemini routing and communications chip.

2.1 Implementation

We have implemented the symmetric and nonsymmetric 2LM methods in C using the PETSc library [1]. We implemented three matrices K, Q and the coarse grid preconditioner P. The matrices P,Q are implemented as PETSc shell matrices while the K matrix is assembled into a `seqaij` matrix. In other words, the matrix K is assembled into PETSc's parallel compressed row storage sparse matrix format, while

the matrices P and Q are not assembled but instead a matrix-vector multiplication routine is provided to PETSc. The matrices P and Q are not assembled because they are not sparse.

We use a PETSc parallel Krylov space solver on (14) or (15) as an "outer iteration". Each step of the outer iteration requires multiplying a given vector by the matrices P, Q, K. The matrix-vector product $K\lambda$ is a straightforward sparse matrix-dense vector product. The matrix-vector product $Q\lambda$ requires solving subdomain problems as per (4). These subdomain problems can in principle become large. Thus, (4) is solved using a PETSc sequential Krylov space solver (ie. a single-processor solver) on (4); this is an "inner iteration" which occurs at each step of the outer iteration. Hence the overall algorithm has an inner-outer iteration structure. In our test implementation, we use a finite difference implementation with a square domain and rectangular subdomains, with one domain assigned per MPI task with affinity to a single core.

2.1.1 The matrix K

The solution λ to the linear systems, (11) or (12) is a multi-valued trace, with one function value per artificial interface point per subdomain. In PETSc, the rows of λ are distributed such that the indices of the same domain are assigned to a single processor,

$$\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_p \end{bmatrix}.$$

Each entry in λ corresponds to an artificial interface grid point. When two or more subdomains are adjacent, then some entries of λ correspond to the same artificial interface point.

Each processor lists the physical grid points on its artificial interface; this information is shared with neighboring subdomains using MPI explicitly. When solving subdomain problems, we work with small-dimensional local vectors. The Robin data λ_j on subdomain Ω_j has length $n_{\Gamma j}$; we write $\lambda_j = (\lambda_i^{(j)})_{i=1}^{n_{\Gamma j}}$. Mapping from the "local index" i to a "global offset" is achieved with the function $F_j(i) = i + \sum_{k<j} n_{\Gamma k}$. The size of the matrix K is $\sum_{k=1}^{p} n_{\Gamma k}$. Given this information, each processor is able to assemble its own rows of K.

2.1.2 The matrix Q

We begin by showing that the matrix-vector product $\lambda_k \mapsto Q_k\lambda_k$ can be computed by solving a local sparse problem. Setting $f = 0$ (and hence $g = 0$) in (4) and (5) shows that $Q_k\lambda_k = au_{\Gamma k}$, where $u_{\Gamma k}$ is defined by,

$$\begin{bmatrix} A_{IIk} & A_{I\Gamma k} \\ A_{\Gamma Ik} & A_{\Gamma\Gamma k} + aI \end{bmatrix} \begin{bmatrix} u_{Ik} \\ u_{\Gamma k} \end{bmatrix} = \begin{bmatrix} 0 \\ \lambda_k \end{bmatrix}. \tag{16}$$

Thus, in order to calculate the matrix-vector product $Q\lambda$, each processor solves the Robin local problem (16) and outputs $Q_k\lambda_k = au_{\Gamma k}$.

The local problem (16) can in principle be solved using eg. a Cholesky decomposition. However, we found that using a Cholesky decomposition leads to large amounts of fill-in and poor performance. Thus, we solve the local problem (16) using the Conjugate Gradient method with relative convergence tolerance 1e-10 and absolute convergence tolerance 1e-9. For the local problem (16), we use the incomplete Cholesky ICC(ℓ) preconditioner [2]. The incomplete Cholesky preconditioner is a compromise between higher fill-in (leading in the limit to a direct solver) and lower fill-in (leading in the limit to a diagonal preconditioner). We found that a "factor level" $\ell = 10$ gives better overall performance for our problem sizes.

2.1.3 The preconditioner P

The coarse grid preconditioner matrix P defined in (13) is in principle an enormous parallel matrix. Nevertheless, we will describe an efficient way to compute the matrix-vector product $\lambda \mapsto P^{-1}\lambda$ efficiently on a single processor (with some global communication). For $j = 1, \ldots, p$ we denote $n_{\Gamma j}$ the number of vertices on the artificial interface $\partial\Omega_j \cap \Gamma$ and we define the matrix $J := \mathrm{diag}(\frac{1}{\sqrt{n_{\Gamma 1}}}\mathbf{1}_{n_{\Gamma 1}}, \ldots, \frac{1}{\sqrt{n_{\Gamma p}}}\mathbf{1}_{n_{\Gamma p}})$ where $\mathbf{1}_j$ denotes the jth dimensional column vector of ones. The columns of J span the "coarse space" of piecewise constant functions, which are constant on each local artificial interface $\Gamma_k = \partial\Omega_k \cap \Gamma$. The coarse space for the preconditioner (13) is the kernel of S, which is contained in the column span of J. Thus, we define $E := JJ^T$ and,

$$P^{-1} := (I - EKE)^{-1} = I - JJ^T - J(\overbrace{J^TKJ - I}^{L})^{-1}J^T.$$

Note that although P^{-1} is dense, we can compute $\lambda \mapsto P^{-1}\lambda$ efficiently, in a matrix-free way, via the formula $P^{-1}\lambda = \lambda - J(J^T\lambda) - J(L^{-1}(J^T\lambda))$.

Given the assembled parallel sparse matrix J and its transpose J^T and the assembled (sparse) local matrix L, the algorithm for computing the matrix-vector product $\lambda \mapsto P^{-1}\lambda$ in a matrix-free way is as follows:

1. Given λ, compute the p-dimensional "coarse" vector $\lambda_c = J^T\lambda$ and collect its entries on a single processor as a sequential vector.
2. Define u_c by solving the local, sparse linear problem $Lu_c = \lambda_c$.
3. Output $P^{-1}\lambda = \lambda - J\lambda_c - Ju_c$. Note that multiplication by J involves broadcasting the small local vectors λ_c and u_c to large parallel vectors $J\lambda_c$ and Ju_c.

Table 1: Iteration counts for S2LM.

# Procs.	Domain size 100^2	300^2	1000^2	3000^2
64	216	409	952	2472
256	173	316	782	1753
1024	144	220	411	1090
4096	-	-	301	665

Table 2: Iteration counts for 2LM.

# Procs.	Domain size 100^2	300^2	1000^2	3000^2	10000^2
64	30	58	114	229	-
256	37	35	72	135	-
1024	47	44	42	76	-
4096	-	-	53	50	82

2.1.4 The outer solve

The implementations of the shell matrices P and Q and the assembly of the sparse matrix K have been described. Building on these base implementations, we further form the shell matrices $\lambda \mapsto (Q-K)\lambda$ (implemented as `QminKmul`) and $\lambda \mapsto (I-2K)(Q-K)\lambda$ (implemented as `Imin2KQminKmul`). The PETSc library enables us to use a variety of different solvers. For the outer iteration we experimented with the Generalized Minimal Residual `KSPGMRES` and the Flexible Generalised Minimal Residual method `KSPFGMRES` on shell matrices `QminKmul` and `Imin2KmulQminK`, with the preconditioner P. For the `KSPFGMRES` solver we set the relative convergence tolerance $1e-7$ and the absolute convergence tolerance $1e-6$.

Recall that GMRES is an iterative method that computes the approximate solution $x_k \in x_0 + \text{span}\{r_0, Ar_0, \ldots A^k r_0\}$ which minimizes the residual norm $\|b-Ax_k\|_2$. The efficient implementation of the least-squares problem relies on the identity

$$AV_k = V_{k+1}\tilde{H}_k, \tag{17}$$

where V_k is an orthonormal basis of the Krylov space and $\tilde{H}_k$. is an upper Hessenberg matrix; cf. [7] for details. The Flexible GMRES algorithm [6] replaces (17) by,

$$AZ_m = V_{k+1}\tilde{H}_k, \tag{18}$$

and allows one to vary the preconditioner at each iteration, which required testing since our matrix-vector products are inexact.

2.1.5 Experiments at large scale

Results for the iteration counts of the S2LM and 2LM methods are presented. In both cases the Flexible GMRES algorithm for the outer solver and the Conjugate Gradient algorithm for the inner solver were used. The preconditioner for the outer solve is the shell matrix P, while the preconditioner for the inner solve is the incomplete Cholesky ICC(10) of (16). The other parameters for the solvers have been specified in sections 2.1.2 and 2.1.4.

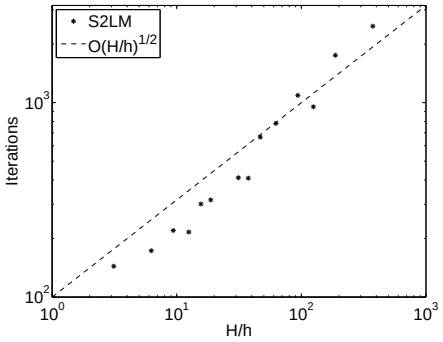

Fig. 1: Scaling of S2LM.

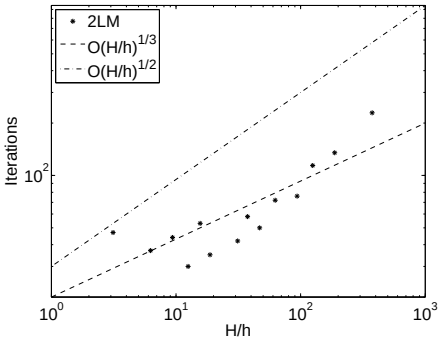

Fig. 2: Scaling of 2LM.

The implementation used here is limited to a square domain in two dimensions using a finite difference discretization. This choice was made entirely for the simplicity of implementation. The domains vary from 100^2 to 10000^2 grid points (and hence the largest problem has 10^8 degrees of freedom). These domains are partitioned into 64 to 4096 subdomains, which again is limited to a square number. This domain decomposition is mapped to the MPI decomposition on HECToR.

The symmetric (11) and nonsymmetric systems (12) are solved, with tolerances as in section 2.1.4; the outer iteration counts are reported in Tables 1 and 2. The computational cost per outer iteration for a fixed domain and subdomain is constant. The inner iterations are not reported as the ICC preconditioner is used for simplicity rather than the optimal multigrid which would be used as first choice in a production implementation. In addition to these raw iteration counts, we also plot the scaling of the methods against the ratio H/h in Figs. 1 and 2.

The S2LM performance is well explained by the condition number estimate of Theorem 3. Indeed, the S2LM matrix is symmetric and indefinite and for such systems, one can show that the number of iterations is bounded by a quantity proportional to the condition number. This bound is only sharp when the spectrum of the matrix is perfectly symmetric about the origin. We find that some of our smaller systems perform slightly better than this theoretical estimate.

The 2LM performance appears to be between $O(H/h)^{1/3}$ and $O(H/h)^{1/2}$. The 2LM matrix is nonsymmetric. For nonsymmetric matrices, the condition number does not necessarily predict the performance of the GMRES algorithm. However, in our case, we find that the condition number explains well the performance of the algorithm and that we further get "Krylov acceleration" – the performance may be almost as good as $O(H/h)^{1/3}$.

3 Conclusions

We have provided a large-scale implementation of the 2-Lagrange multiplier methods with cross points and a coarse grid correction, which we have tested on the HECToR supercomputer. Our experiments confirm the good scaling properties of the 2-Lagrange multiplier methods. In the future, we intend to improve our implementation to further explore the scaling to the largest systems.

Acknowledgements We gratefully acknowledge the support of the Centre for Numerical Algorithms and Intelligent Software (EPSRC EP/G036136/1). This work made use of the facilities of HECToR, the UK's national high-performance computing service, which is provided by UoE HPCx Ltd. at the University of Edinburgh, Cray Inc and NAG Ltd., and funded by the Office of Science and Technology through EPSRC's High End Computing Programme.

References

1. Balay, S., Brown, J., Buschelman, K., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc Web page (2012). Http://www.mcs.anl.gov/petsc
2. Chan, T.F., Van Der Vorst, H.A.: Approximate and incomplete factorizations. Parallel Numerical Algorithms, ICASE/LaRC Interdisciplinary Series in Science and Engeneering pp. 167–202 (1997)
3. Drury, S.W., Loisel, S.: Sharp condition number estimates for the symmetric 2-lagrange multiplier method. DD20 proceedings (2011)
4. Farhat, C., Roux, F.X.: A method of finite element tearing and interconnecting and its parallel solution algorithm. Internat. J. Numer. Methods Engrg **32**, pp. 1205–1227 (1991)
5. Loisel, S.: Condition number estimates for the nonoverlapping optimized schwarz method and the 2-lagrange multiplier method for general domains and cross points. To appear in SIAM Journal on Numerical Analysis (2013)
6. Saad, Y.: A flexible inner-outer preconditioned gmres algorithm. SIAM Journal on Scientific Computing **14**, 461–469 (1993)
7. Saad, Y., Schultz, M.: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM Journal on scientific and statistical computing **7**(3), 856–869 (1986)
8. Toselli, A., Widlund, O.B.: Domain Decomposition Methods – Algorithms and Theory, vol. volume 34 of Springer Series in Computational Mathematics. Springer Berlin Heidelberg (2005)